Bucket-List

DOCUMENTATION TECHNIQUE

Application web de gestion de notes et checklists

Framework	Symfony 8 — PHP 8.3.14
Base de données	MySQL avec Doctrine ORM
Moteur de templates	Twig + Asset Mapper
Type de projet	Projet BTS SIO SLAM — 2e année
Auteur	Projet éducatif open-source (GitHub)
Date	Avril – Mai 2026

Table of Contents

1. Introduction

1.1 Présentation du projet

Bucket-List est une application web de gestion personnelle de notes et de checklists. Elle a été développée dans le cadre de la deuxième année de BTS SIO option SLAM, avec pour objectif principal la découverte et la maîtrise du framework PHP Symfony.

L'interface s'inspire volontairement de l'outil Notion : minimaliste, sobre, avec une sidebar de navigation permanente affichant en temps réel toutes les notes et checklists de l'utilisateur. Avec le temps, l'application a évolué pour convenir aussi bien à un usage personnel que professionnel, et est aujourd'hui disponible en open-source sur GitHub.

1.2 Fonctionnalités principales

- Inscription et connexion sécurisée (email + mot de passe haché)
- Création, lecture, modification et archivage de notes
- Création, lecture, modification et archivage de checklists avec items cochables
- Recherche globale dans les notes et checklists
- Gestion complète du profil : pseudo, photo de profil, mot de passe
- Thème clair / sombre persisté par utilisateur en base de données
- Support multilingue (français / anglais) via le composant Translation

1.3 Stack technique

Technologie	Version	Rôle
PHP	8.4.1	Langage serveur
Symfony	8 (flex)	Framework MVC principal
Doctrine ORM	3.x	Accès base de données
MySQL	8.x	SGBD relationnel
Twig	3.x	Moteur de templates
Asset Mapper	-	Gestion des assets (CSS/JS)
JavaScript (Vanilla)	ES6+	Interactivité côté client

2. Installation & Mise en route

2.1 Prérequis

- PHP 8.2 minimum (8.3.14 recommandé)
- Composer (gestionnaire de dépendances PHP)
- Symfony CLI (pour le serveur de développement)
- MySQL 8+ ou MariaDB
- Node.js (pour Asset Mapper, optionnel)

2.2 Commandes d'installation

Créer le projet Symfony (avec le squelette complet) :

```
symfony new bucket-list --webapp
```

Installer les bundles nécessaires un par un :

```
composer require symfony/security-bundle
composer require symfony/form
composer require symfony/validator
composer require symfony/orm-pack
composer require symfony/mailer
composer require symfony/translation
composer require symfony/intl
composer require vich/uploader-bundle
composer require symfonycasts/reset-password-bundle
composer require symfony/ux-turbo
composer require twig/extra-bundle
composer require --dev symfony/maker-bundle
```

2.3 Configuration de la base de données

Dans le fichier .env, configurer la connexion MySQL :

```
DATABASE_URL="mysql://root:@127.0.0.1:3306/bucket_list?serverVersion=8.0"
```

Créer la base de données :

```
php bin/console doctrine:database:create
```

Générer et exécuter la migration (crée toutes les tables) :

```
php bin/console make:migration
php bin/console doctrine:migrations:migrate
```

2.4 Lancement du serveur de développement

```
symfony serve
```

L'application est accessible sur : http://127.0.0.1:8000

2.5 Gestion des assets

Le projet utilise Symfony Asset Mapper (pas de Webpack ni de npm build). Les fichiers CSS et JS du dossier assets/ sont servis directement et versionnés automatiquement :

```
php bin/console asset-map:compile   # production
php bin/console debug:asset-map     # vérifier les chemins
```

3. Architecture du projet

3.1 Structure des dossiers

Dossier / Fichier	Description
src/Controller/	5 contrôleurs : Home, Auth, Note, Checklist, Profile
src/Entity/	4 entités Doctrine : User, Note, Checklist, ChecklistItem
src/Form/	5 types de formulaires Symfony
src/Repository/	4 repositories avec requêtes DQL personnalisées
src/EventSubscriber/	LocaleSubscriber : applique la langue à chaque requête
templates/	Templates Twig (base.html.twig + sous-dossiers par fonctionnalité)
assets/	CSS (app.css, auth.css, dashboard.css) + JS (theme, sidebar, checklist)
config/	Configuration Symfony (security.yaml, services.yaml, packages/...)
migrations/	Migration Doctrine : génère les tables SQL
public/	Point d'entrée (index.php) + assets compilés

3.2 Patron de conception MVC

Symfony suit le patron Modèle-Vue-Contrôleur (MVC) :

- Modèle : les Entités Doctrine + leurs Repositories (src/Entity/, src/Repository/)
- Vue : les templates Twig (templates/)
- Contrôleur : les classes dans src/Controller/ — reçoivent la requête HTTP, interrogent le modèle, retournent une réponse Twig

3.3 Pattern Repository

Chaque entité possède un Repository qui centralise toutes les requêtes en base. Les contrôleurs ne font jamais de requêtes directement : ils appellent les méthodes du Repository.

Exemple de méthode personnalisée dans NoteRepository :

```
public function rechercherNotes(User $user, string $motCle): array
{
    $terme = '%' . $motCle . '%';
    return $this->createQueryBuilder('n')
        ->andWhere('n.user = :user')
        ->setParameter('user', $user)
        ->andWhere('n.titre LIKE :terme OR n.contenu LIKE :terme')
        ->setParameter('terme', $terme)
        ->orderBy('n.createdAt', 'DESC')
```

```
            ->getQuery()->getResult();
    }
```

4. Base de données

4.1 Schéma relationnel

La base de données comprend 5 tables créées par la migration Doctrine :

Table	Clé primaire	Clé étrangère	Champs principaux
user	id (INT)	—	email, password, pseudo, photo_filename, theme, langue, roles (JSON)
note	id (INT)	user_id → user.id	titre, contenu (TEXT), is_archived, created_at, updated_at
checklist	id (INT)	user_id → user.id	titre, is_archived, created_at, updated_at
checklist_item	id (INT)	checklist_id → checklist.id	texte, is_checked, created_at
messenger_messages	id (BIGINT)	—	Gestion asynchrone Symfony Messenger (généré automatiquement)

4.2 Relations entre entités

- User → Note : OneToMany (un utilisateur possède plusieurs notes)
- User → Checklist : OneToMany (un utilisateur possède plusieurs checklists)
- Checklist → ChecklistItem : OneToMany avec cascade remove + orphanRemoval

4.3 Lifecycle Callbacks Doctrine

Les dates created_at et updated_at ne sont jamais remplies manuellement. Doctrine les gère automatiquement via des hooks déclarés dans les entités :

```
#[ORM\HasLifecycleCallbacks]  // indispensable sur la classe
#[ORM\PrePersist]
public function onPrePersist(): void  // appelé avant INSERT
{
    $this->createdAt = new \DateTime();
    $this->updatedAt = new \DateTime();
}
```

```
#[ORM\PreUpdate]
public function onPreUpdate(): void  // appelé avant UPDATE
{
    $this->updatedAt = new \DateTime();
}
```

4.4 Migration SQL

La migration (Version20260409210546) a été générée automatiquement par la commande make:migration. Elle contient le SQL de création de toutes les tables :

```
CREATE TABLE `user` (
    id INT AUTO_INCREMENT NOT NULL,
    email VARCHAR(180) NOT NULL,
    roles JSON NOT NULL,
    password VARCHAR(255) NOT NULL,
    pseudo VARCHAR(50) NOT NULL,
    photo_filename VARCHAR(255) DEFAULT NULL,
    theme VARCHAR(10) DEFAULT 'light' NOT NULL,
    langue VARCHAR(5) DEFAULT 'fr' NOT NULL,
    UNIQUE INDEX UNIQ_8D93D649E7927C74 (email),
    PRIMARY KEY (id)
) DEFAULT CHARACTER SET utf8;
```

5. Les Entités Doctrine

5.1 Entité User

L'entité User est la plus complexe : elle implémente deux interfaces Symfony imposées par le système de sécurité.

```
class User implements UserInterface, PasswordAuthenticatedUserInterface
```

Champ	Type ORM	Description
id	INT (auto)	Clé primaire auto-incrémentée
email	VARCHAR(180)	Identifiant de connexion, unique en base
password	VARCHAR(255)	Mot de passe haché (bcrypt/argon2 via 'auto')
roles	JSON	Tableau de rôles, ROLE_USER toujours présent
pseudo	VARCHAR(50)	Nom d'affichage dans l'interface
photo_filename	VARCHAR(255) null	Nom du fichier avatar (stocké dans public/uploads/avatars/)
theme	VARCHAR(10)	'light' ou 'dark', défaut 'light'
langue	VARCHAR(5)	'fr' ou 'en', défaut 'fr'

5.2 Entité Note

Une note appartient à un utilisateur (ManyToOne), possède un titre obligatoire et un contenu texte libre optionnel.

```
#[ORM\ManyToOne(inversedBy: 'notes')]
#[ORM\JoinColumn(nullable: false)]
private ?User $user = null;
```

5.3 Entité Checklist + ChecklistItem

La Checklist est un conteneur de tâches. Elle contient une collection d'items via une relation OneToMany :

```
#[ORM\OneToMany(
    mappedBy: 'checklist',
    targetEntity: ChecklistItem::class,
    cascade: ['persist', 'remove'],
    orphanRemoval: true
)]
#[ORM\OrderBy(['createdAt' => 'DESC'])]
private Collection $items;
```

La méthode utilitaire getNombreItemsCocheches() permet d'afficher la progression (ex : 3/5 tâches) :

```
public function getNombreItemsCocheches(): int
{
    $count = 0;
    foreach ($this->items as $item) {
        if ($item->isChecked()) { $count++; }
    }
    return $count;
}
```

6. Les Contrôleurs

6.1 Vue d'ensemble des routes

Route	Méthode HTTP	Nom Symfony	Description
/	GET	app_home	Page d'accueil publique
/login	GET/POST	app_login	Page de connexion
/register	GET/POST	app_register	Formulaire d'inscription
/logout	GET	app_logout	Déconnexion (Symfony)
/dashboard	GET	app_dashboard	Tableau de bord
/note/creer	GET/POST	app_note_create	Créer une note
/note/{id}	GET	app_note_read	Lire une note
/note/{id}/modifier	GET/POST	app_note_update	Modifier une note
/note/{id}/archiver	POST	app_note_archive	Archiver une note
/recherche	GET	app_search	Recherche globale
/checklist/creer	GET/POST	app_checklist_create	Créer une checklist
/checklist/{id}	GET	app_checklist_read	Lire une checklist
/checklist/{id}/modifier	GET/POST	app_checklist_update	Modifier une checklist
/checklist/{id}/archiver	POST	app_checklist_archive	Archiver une checklist
/checklist/item/{id}/cocher	POST	app_checklist_item_toggle	Cocher/décocher un item
/profil	GET/POST	app_profile	Profil utilisateur

6.2 Sécurité des contrôleurs — IsGranted

Tous les contrôleurs sauf HomeController et AuthController sont protégés par l'attribut PHP placé au niveau de la classe :

```
#[IsGranted('ROLE_USER')]
class NoteController extends AbstractController
```

Cela signifie que toutes les méthodes de ce contrôleur exigent d'être connecté. Symfony redirige automatiquement vers /login si la session est absente.

De plus, chaque action vérifie que la ressource appartient bien à l'utilisateur connecté (ownership check) :

```
if ($note->getUser() !== $this->getUser()) {
    throw $this->createAccessDeniedException('Vous n\'avez pas accès à cette note.');
```

```
}
```

6.3 ParamConverter Symfony

Quand une route contient un paramètre comme {id}, Symfony est capable de récupérer automatiquement l'objet Doctrine correspondant :

```
// Symfony convertit automatiquement l'id en objet Note
#[Route('/note/{id}', name: 'app_note_read', requirements: ['id' => '\d+'])]
public function read(Note $note, NoteRepository $noteRepository): Response
```

6.4 Cycle de vie d'un formulaire Symfony

Chaque action de création/modification suit toujours le même pattern :

```
$formulaire = $this->createForm(NoteType::class, $note);  // Créer
$formulaire->handleRequest($request);                      // Lier la requête
```

```
if ($formulaire->isSubmitted() && $formulaire->isValid()) {
    // Traitement : persist + flush ou flush seul si entité existante
    $em->persist($note);  // seulement pour une nouvelle entité
    $em->flush();          // écrit en base
    return $this->redirectToRoute('...');
}
```

7. Système d'authentification

7.1 Configuration security.yaml

Tout le système d'authentification est déclaré dans config/packages/security.yaml :

```
password_hashers:
    App\Entity\User:
        algorithm: auto   # bcrypt ou argon2 selon la config serveur
```

```
providers:
    app_user_provider:
        entity:
            class: App\Entity\User
            property: email  # identifiant de connexion
```

```
firewalls:
    main:
        form_login:
            login_path: app_login
            check_path: app_login
            default_target_path: app_dashboard
        logout:
            path: app_logout
            target: app_home
```

```
access_control:
    - { path: ^/$, roles: PUBLIC_ACCESS }
    - { path: ^/login, roles: PUBLIC_ACCESS }
    - { path: ^/register, roles: PUBLIC_ACCESS }
    - { path: ^/, roles: ROLE_USER }
```

7.2 Inscription (Register)

L'inscription utilise RegisterType, qui contient un champ plainPassword mapped: false (ne correspond à aucun champ de l'entité User). Le mot de passe en clair est récupéré depuis le formulaire, haché puis sauvegardé :

```
$motDePasseClair = $formulaire->get('plainPassword')->getData();
$motDePasseHache = $passwordHasher->hashPassword($user, $motDePasseClair);
$user->setPassword($motDePasseHache);
```

7.3 Connexion (Login)

La connexion est entièrement gérée par Symfony Security. La méthode login() dans AuthController ne traite pas elle-même le formulaire : elle fournit juste les données d'erreur et le dernier email saisi pour pré-remplir le champ.

```
$erreur = $authenticationUtils->getLastAuthenticationError();
$dernierEmail = $authenticationUtils->getLastUsername();
```

7.4 Déconnexion

La déconnexion est entièrement interceptée par Symfony avant d'atteindre le contrôleur. La méthode logout() dans AuthController ne sera jamais appelée. Elle lève une LogicException par convention :

```
#[Route('/logout', name: 'app_logout')]
public function logout(): void
{
    throw new \LogicException('Jamais appelée : Symfony l\'intercepte.');
}
```

8. Fonctionnalités avancées

8.1 Système de thème clair / sombre

Le thème est stocké directement sur l'entité User en base de données (champ theme = 'light' ou 'dark'). Il est appliqué côté serveur dans base.html.twig :

```
{# Twig — appliqué sur le body #}
<body class="theme-{{ app.user.theme }}">
```

Le fichier assets/js/theme.js permet une bascule instantanée en JavaScript (sans rechargement de page) :

```
if (body.classList.contains('theme-dark')) {
    body.classList.replace('theme-dark', 'theme-light');
} else {
    body.classList.replace('theme-light', 'theme-dark');
}
```

Le choix est ensuite persisté via le formulaire de profil (ProfileController) lors de la prochaine sauvegarde.

8.2 Système multilingue (fr/en)

La langue est gérée à trois niveaux :

- Stockage : champ langue dans l'entité User (valeur 'fr' ou 'en')
- Application par requête : le LocaleSubscriber lit la langue de l'utilisateur à chaque requête HTTP
- Changement immédiat : après sauvegarde du profil, la session est mise à jour sans attendre la prochaine connexion

Le LocaleSubscriber implémente EventSubscriberInterface et s'abonne à KernelEvents::REQUEST avec une priorité de 20 pour s'exécuter avant le routeur :

```
public static function getSubscribedEvents(): array
{
    return [
        KernelEvents::REQUEST => [['onKernelRequest', 20]],
    ];
}
```

8.3 Recherche globale

La route /recherche?q=mot-clé permet de chercher dans les notes ET les checklists simultanément. La recherche utilise l'opérateur SQL LIKE avec des wildcards :

```
// Dans NoteRepository
```

```
->andWhere('n.titre LIKE :terme OR n.contenu LIKE :terme')
->setParameter('terme', '%' . $motCle . '%')
```

8.4 Archivage des contenus

L'archivage est une suppression douce (soft delete) : la note ou checklist reste en base mais son champ is_archived passe à true. Elle disparaît de la liste principale et apparaît dans une section « Archives » de la sidebar.

```
#[Route('/note/{id}/archiver', methods: ['POST'])]
public function archive(Note $note, EntityManagerInterface $em): Response
{
    $note->setIsArchived(true);
    $em->flush();
    return $this->redirectToRoute('app_dashboard');
}
```

8.5 Gestion de la photo de profil

L'upload de photo ne passe pas par VichUploaderBundle pour la sauvegarde finale (ce bundle est installé mais l'upload est géré manuellement dans ProfileController) :

```
$photoFile = $formulaire->get('photoFile')->getData();
if ($photoFile) {
    $nouveauNom = uniqid('avatar_') . '.' . $photoFile->guessExtension();
    $photoFile->move(
        $this->getParameter('photos_directory'), // config/services.yaml
        $nouveauNom
    );
    $user->setPhotoFilename($nouveauNom);
}
```

Le paramètre photos_directory est défini dans config/services.yaml :

```
parameters:
    photos_directory: '%kernel.project_dir%/public/uploads/avatars'
```

8.6 Gestion dynamique des items de checklist (JavaScript)

Le formulaire de création/modification de checklist permet d'ajouter et supprimer des items dynamiquement sans rechargement de page. Cela repose sur le mécanisme de prototype Symfony :

- Symfony génère un attribut data-prototype sur le conteneur contenant le HTML d'un input vide
- Le placeholder __name__ est remplacé par l'index réel au moment de l'injection dans le DOM
- Un bouton ✕ est ajouté à chaque ligne pour supprimer l'item du DOM

```
// Remplacement du placeholder par l'index
```

```
var inputHtml = prototype.replace(/__name__/g, index);
conteneur.dataset.index = index + 1; // incrémentation pour le suivant
```

9. Les Formulaires Symfony

9.1 Liste des types de formulaires

Classe	Description
RegisterType	Champs : email, pseudo, plainPassword (mapped: false)
NoteType	Champs : titre (TextType), contenu (TextareaType)
ChecklistType	Champs : titre + CollectionType d'items (ChecklistItemType)
ChecklistItemType	Champ unique : texte (TextType)
ProfileType	Champs : pseudo, email, photoFile (FileType, mapped: false), currentPassword, newPassword, theme (ChoiceType), langue (ChoiceType)

9.2 Formulaire imbriqué — ChecklistType avec CollectionType

C'est la partie la plus complexe du projet. ChecklistType utilise un CollectionType pour gérer une collection de sous-formulaires ChecklistItemType :

```
$builder->add('items', CollectionType::class, [
    'entry_type'    => ChecklistItemType::class,
    'allow_add'     => true,   // JavaScript peut ajouter des éléments
    'allow_delete'  => true,   // JavaScript peut en retirer
    'by_reference'  => false,  // force l'utilisation de addItem()/removeItem()
    'attr'          => ['data-prototype' => ..., 'data-index' => 0],
]);
```

10. Front-end & Assets

10.1 Symfony Asset Mapper

Le projet n'utilise pas Webpack ni npm build. Symfony Asset Mapper sert les fichiers CSS/JS directement depuis le dossier assets/ en les versionnant automatiquement (hash dans le nom de fichier compilé).

```
// importmap.php — déclare les dépendances
// app.js — point d'entrée principal
// assets/styles/app.css — feuille de styles globale
```

10.2 Fichiers CSS

Fichier	Contenu
assets/styles/app.css	Styles globaux : variables CSS, sidebar, notes, thème clair/sombre, responsive
assets/styles/auth.css	Styles spécifiques aux pages login et register
assets/styles/dashboard.css	Styles du tableau de bord principal

10.3 Fichiers JavaScript

Fichier	Rôle
assets/js/theme.js	Bascule thème clair/sombre côté client (sans rechargement)
assets/js/sidebar.js	Comportement de la sidebar sur mobile (menu burger)
assets/js/checklist.js	Ajout/suppression dynamique d'items de checklist (prototype Symfony)

10.4 Templates Twig

L'héritage Twig est organisé autour d'un template de base :

- base.html.twig : layout global (head, body, imports CSS/JS, inclusion de la sidebar)
- partials/_sidebar.html.twig : sidebar avec navigation, liste des notes/checklists, barre de recherche
- home/index.html.twig : page d'accueil publique
- auth/login.html.twig et auth/register.html.twig : formulaires d'authentification
- note/ : create_note, read_note, update_note, search (4 templates)
- checklist/ : create_checklist, read_checklist, update_checklist (3 templates)
- profile/index.html.twig : page profil + paramètres

11. Sécurité

11.1 Mesures de sécurité en place

Mécanisme	Mise en œuvre
Hachage des mots de passe	Algorithme 'auto' (bcrypt/argon2) via UserPasswordHasherInterface
Authentification	Système de sessions Symfony Security (firewall main)
Contrôle d'accès global	access_control dans security.yaml : tout ^ / requiert ROLE_USER
Contrôle d'accès par ressource	createAccessDeniedException() si la ressource n'appartient pas à l'utilisateur
Protection CSRF	Tokens CSRF Symfony activés sur tous les formulaires POST
Validation des données	Contraintes Assert sur les entités (NotBlank, Length, Email)
Actions destructrices en POST	Archive et toggle item en methods: ['POST'] pour éviter les requêtes GET non intentionnelles

11.2 Validation Symfony

Les contraintes de validation sont déclarées directement sur les propriétés des entités avec des attributs PHP :

```
#[Assert\NotBlank(message: 'Le titre est obligatoire.')]
#[Assert\Length(max: 255, maxMessage: 'Max 255 caractères.')]
private ?string $titre = null;
```

Symfony vérifie ces contraintes automatiquement lors de $formulaire->isValid(). En cas d'erreur, les messages sont retournés directement dans le template Twig.

12. Problèmes rencontrés & Solutions

12.1 Assets non trouvés (CSS/JS introuvables)

Problème : Après installation, les dossiers public/assets/js/ et public/assets/css/ n'existaient pas. L'application était chargée sans style.

Cause : Asset Mapper n'avait pas encore compilé les assets en mode développement.

Solution : Vider le cache et relancer le serveur Symfony. En développement, Asset Mapper sert les fichiers à la volée.

```
php bin/console cache:clear

symfony serve
```

12.2 Conflit de routes /note/creer vs /note/{id}

Problème : La route /note/creer entrait en conflit avec /note/{id} car Symfony essayait de convertir 'creer' en entier pour le ParamConverter.

Solution : Ajouter un requirements sur la route {id} pour forcer un entier, et placer la route /creer AVANT la route /{id} dans le contrôleur.

```
#[Route('/note/{id}', requirements: ['id' => '\d+'])]
```

12.3 Changement de langue sans effet immédiat

Problème : Après avoir sauvegardé la nouvelle langue dans le profil, la langue ne changeait qu'à la prochaine connexion.

Solution : Mettre à jour la session immédiatement après la sauvegarde dans ProfileController :

```
$request->getSession()->set('_locale', $user->getLangue());
```

12.4 Table 'user' — mot-clé réservé SQL

Problème : MySQL refusait la création de la table 'user' car c'est un mot-clé réservé SQL.

Solution : Encapsuler le nom de la table avec des backticks dans l'annotation Doctrine :

```
#[ORM\Table(name: '`user`')]
```

12.5 Items de checklist non supprimés

Problème : Lors de la modification d'une checklist, supprimer un item du formulaire ne le supprimait pas en base.

Solution : Ajouter orphanRemoval: true sur la relation OneToMany + by_reference: false dans le CollectionType pour forcer l'appel à removeItem().

13. Commandes utilisées

13.1 Création et configuration

Commande	Description
symfony new bucket-list --webapp	Créer le projet avec le squelette complet
composer require <bundle>	Installer un bundle tiers
symfony serve	Lancer le serveur de développement
symfony serve --port=8002	Lancer sur un port spécifique

13.2 Base de données (Doctrine)

Commande	Description
php bin/console doctrine:database:create	Créer la base de données
php bin/console make:migration	Générer le fichier de migration SQL
php bin/console doctrine:migrations:migrate	Exécuter les migrations en attente
php bin/console doctrine:schema:validate	Vérifier la cohérence schéma/entités

13.3 Génération de code (Maker Bundle)

Commande	Description
php bin/console make:entity	Créer ou modifier une entité et son repository
php bin/console make:controller	Créer un contrôleur vide
php bin/console make:form	Générer un type de formulaire
php bin/console make:user	Créer l'entité User avec les interfaces de sécurité
php bin/console make:security:form-login	Générer le système de connexion

13.4 Débogage

Commande	Description
php bin/console debug:router	Lister toutes les routes de l'application
php bin/console debug:container	Lister tous les services disponibles
php bin/console debug:asset-map	Lister les assets mappés
php bin/console cache:clear	Vider le cache Symfony
php bin/console debug:autowiring	Voir les services injectables disponibles

14. Bilan & Perspectives

14.1 Compétences BTS SIO mises en œuvre

Activité	Mise en œuvre dans Bucket-List
B2.1 — Développement applicatif	Architecture MVC, entités, contrôleurs, formulaires, repositories
B2.1 — Exploitation du framework	Symfony 8 : Security, Doctrine ORM, Twig, Form, Translation, EventSubscriber
B2.2 — Maintenance corrective	Corrections des bugs routes, assets, langue, items checklist
B2.3 — Gestion des données	Modélisation BDD, migrations Doctrine, requêtes DQL personnalisées
Sécurité des données	Hachage mot de passe, CSRF, contrôle accès par ressource, validation

14.2 Points forts du projet

- Architecture claire respectant les bonnes pratiques Symfony
- Code entièrement commenté pour faciliter la compréhension et la maintenance
- Séparation des responsabilités : contrôleur, entité, repository, formulaire, template
- Sécurité bien pensée : aucune ressource accessible sans vérification d'appartenance
- Interface fluide inspirée de Notion, thème clair/sombre persisté
- Disponible en open-source sur GitHub

14.3 Pistes d'amélioration

- Ajouter une fonctionnalité de désarchivage (restore) en plus de l'archivage
- Implémenter la suppression définitive avec confirmation
- Ajouter des tests unitaires (PHPUnit) et fonctionnels
- Intégrer une API REST pour une application mobile
- Ajouter un système de tags/étiquettes sur les notes
- Mise en production avec Docker + déploiement CI/CD

Documentation rédigée dans le cadre du BTS SIO option SLAM — 2e année